
9.6 Interrupt Tasks and Interrupt Procedures

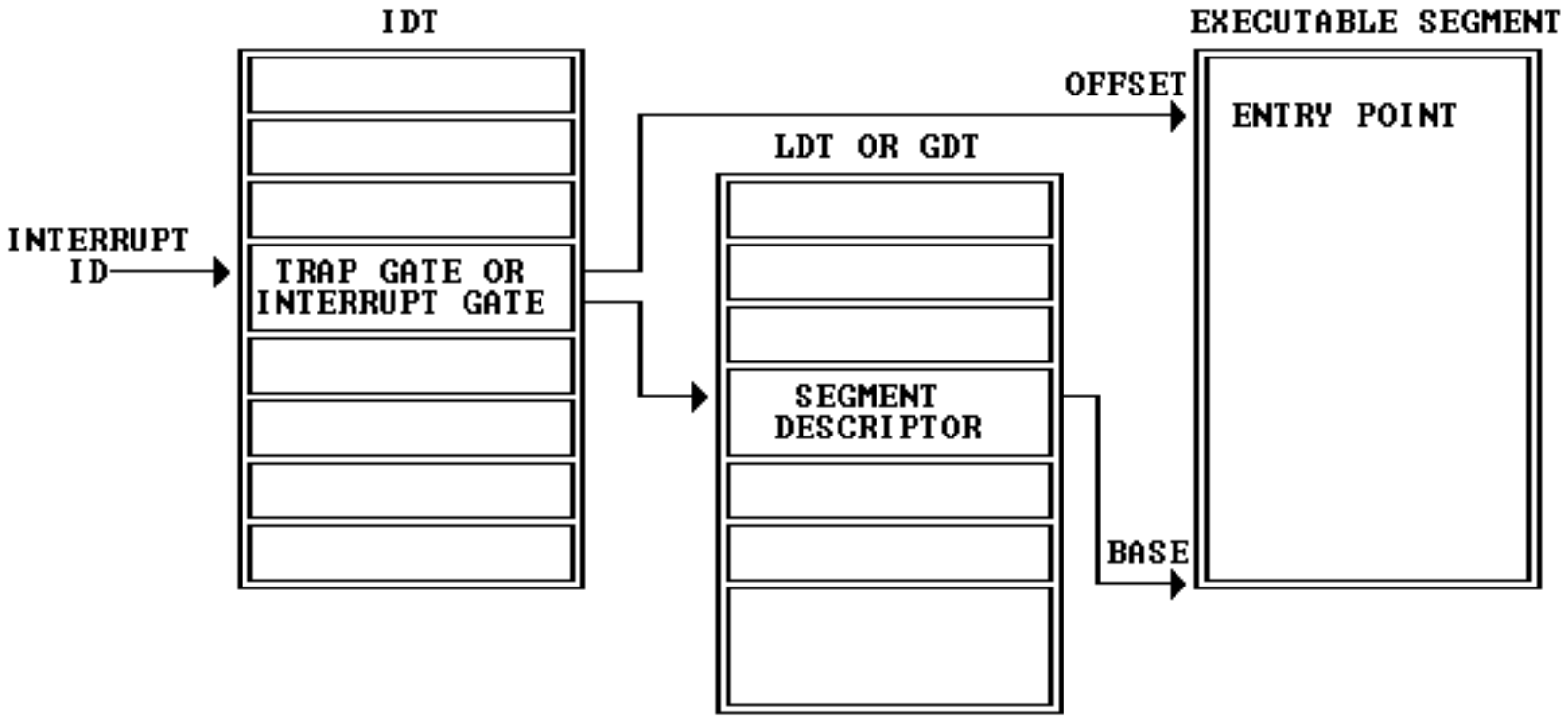
Just as a CALL instruction can call either a procedure or a task, so an interrupt or exception can "call" an interrupt handler that is either a procedure or a task. When responding to an interrupt or exception, the processor uses the interrupt or exception identifier to index a descriptor in the IDT. If the processor indexes to an interrupt gate or trap gate, it invokes the handler in a manner similar to a CALL to a call gate. If the processor finds a task gate, it causes a task switch in a manner similar to a CALL to a task gate.

9.6.1 Interrupt Procedures

An interrupt gate or trap gate points indirectly to a procedure which will execute in the context of the currently executing task as illustrated by Figure 9-4 . The selector of the gate points to an executable-segment descriptor in either the GDT or the current LDT. The offset field of the gate points to the beginning of the interrupt or exception handling procedure.

The 80386 invokes an interrupt or exception handling procedure in much the same manner as it CALLs a procedure; the differences are explained in the following sections.

Figure 9-4. Interrupt Vectoring for Procedures



9.6.1.1 Stack of Interrupt Procedure

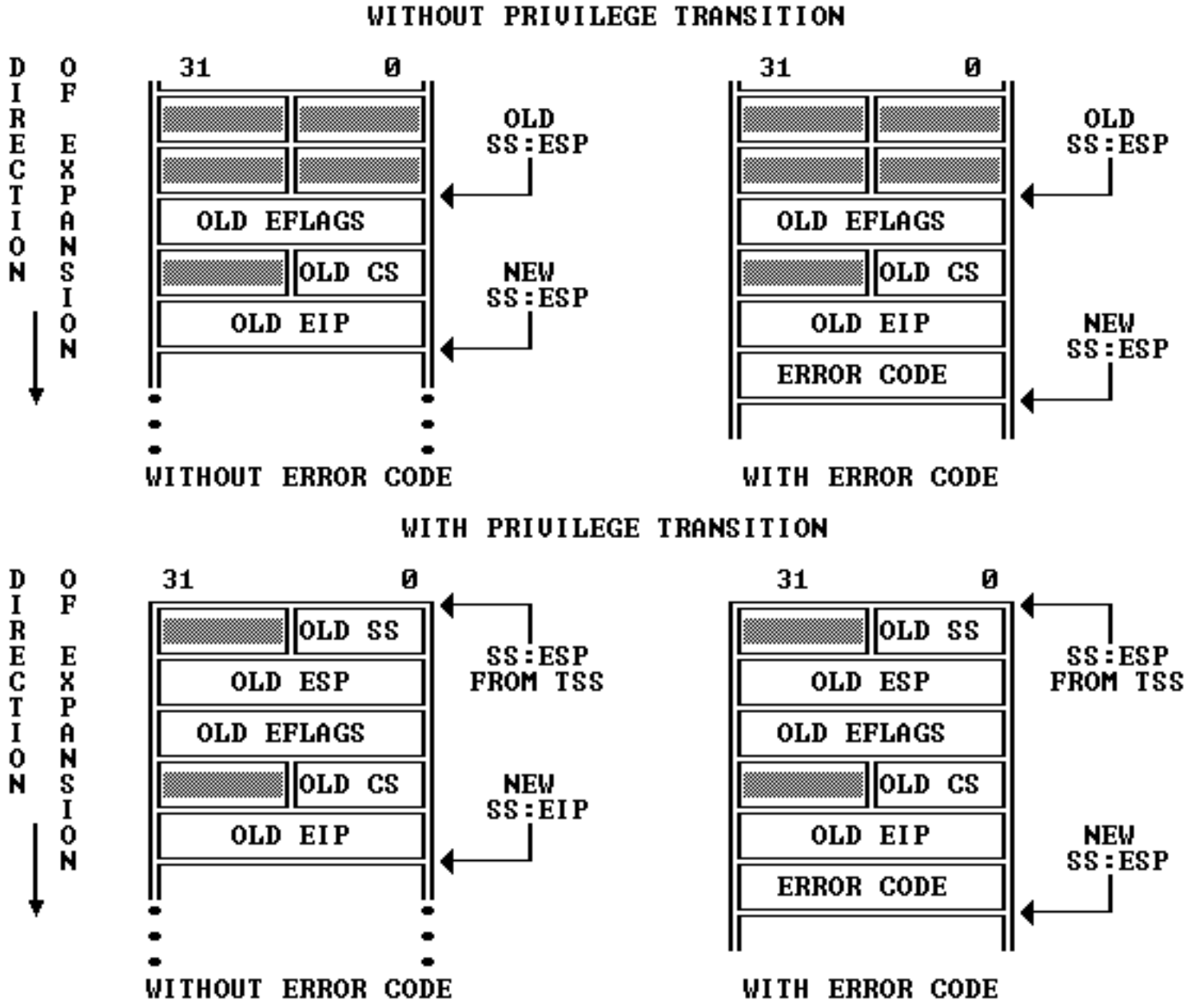
Just as with a control transfer due to a CALL instruction, a control transfer to an interrupt or exception handling procedure uses the stack to store the information needed for returning to the original procedure. As Figure 9-5 shows, an interrupt pushes the EFLAGS register onto the stack before the pointer to the interrupted instruction.

Certain types of exceptions also cause an error code to be pushed on the stack. An exception handler can use the error code to help diagnose the exception.

9.6.1.2 Returning from an Interrupt Procedure

An interrupt procedure also differs from a normal procedure in the method of leaving the procedure. The IRET instruction is used to exit from an interrupt procedure. IRET is similar to RET except that IRET increments ESP by an extra four bytes (because of the flags on the stack) and moves the saved flags into the EFLAGS register. The IOPL field of EFLAGS is changed only if the CPL is zero. The IF flag is changed only if CPL <= IOPL.

Figure 9-5. Stack Layout after Exception of Interrupt



9.6.1.3 Flags Usage by Interrupt Procedure

Interrupts that vector through either interrupt gates or trap gates cause TF (the trap flag) to be reset after the current value of TF is saved on the stack as part of EFLAGS. By this action the processor prevents debugging activity that uses single-stepping from affecting interrupt response. A subsequent IRET instruction restores TF to the value in the EFLAGS image on the stack.

The difference between an interrupt gate and a trap gate is in the effect on IF (the interrupt-enable flag). An interrupt that vectors through an interrupt gate resets IF, thereby preventing other interrupts from interfering with the current interrupt handler. A subsequent IRET instruction restores IF to the value in the EFLAGS image on the stack. An interrupt through a trap gate does not change IF.

9.6.1.4 Protection in Interrupt Procedures

The privilege rule that governs interrupt procedures is similar to that for procedure calls: the CPU does not permit an interrupt to transfer control to a procedure in a segment of lesser privilege (numerically greater privilege level) than the current privilege level. An attempt to violate this rule results in a general protection exception.

Because occurrence of interrupts is not generally predictable, this privilege rule effectively imposes restrictions on the privilege levels at which interrupt and exception handling procedures can execute. Either of the following strategies can be employed to ensure that the privilege rule is never violated.

- Place the handler in a conforming segment. This strategy suits the handlers for certain exceptions (divide error, for example). Such a handler must use only the data available to it from the stack. If it needed data from a data segment, the data segment would have to have privilege level three, thereby making it unprotected.
- Place the handler procedure in a privilege level zero segment.

9.6.2 Interrupt Tasks

A task gate in the IDT points indirectly to a task, as Figure 9-6 illustrates. The selector of the gate points to a TSS descriptor in the GDT.

When an interrupt or exception vectors to a task gate in the IDT, a task switch results. Handling an interrupt with a separate task offers two advantages:

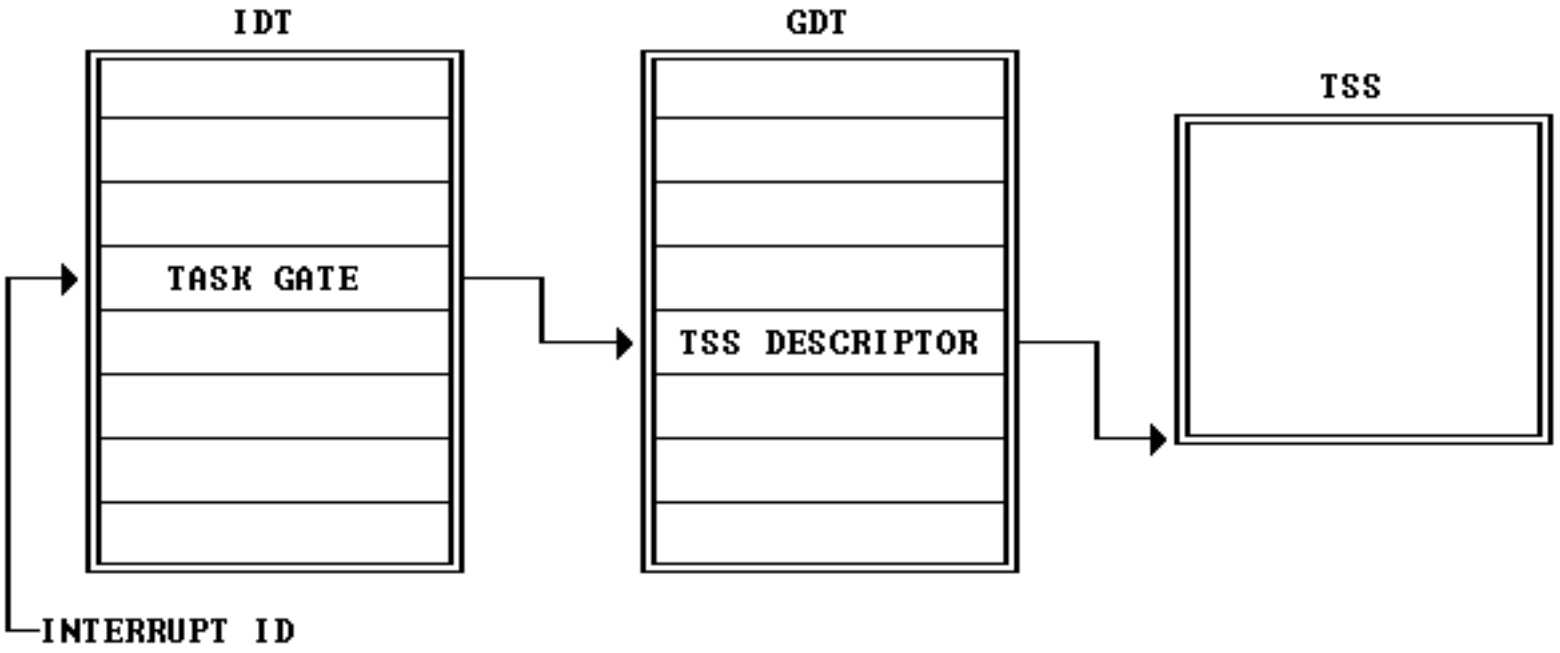
- The entire context is saved automatically.
- The interrupt handler can be isolated from other tasks by giving it a separate address space, either via its LDT or via its page directory.

The actions that the processor takes to perform a task switch are discussed in Chapter 7. The interrupt task returns to the interrupted task by executing an IRET instruction.

If the task switch is caused by an exception that has an error code, the processor automatically pushes the error code onto the stack that corresponds to the privilege level of the first instruction to be executed in the interrupt task.

When interrupt tasks are used in an operating system for the 80386, there are actually two schedulers: the software scheduler (part of the operating system) and the hardware scheduler (part of the processor's interrupt mechanism). The design of the software scheduler should account for the fact that the hardware scheduler may dispatch an interrupt task whenever interrupts are enabled.

Figure 9-6. Interrupt Vectoring for Tasks



---

up: Chapter 9 -- Exceptions and Interrupts
prev: 9.5 IDT Descriptors
next: 9.7 Error Code